Accessing Enterprise JavaBeans Using JNDI in SAP NetWeaver Application Server, Java ™ EE 5 Edition

Applies to:

SAP NetWeaver Application Server, Java ™ EE 5 Edition

Summary

Enterprise JavaBeans (EJB) can be accessed using the Java Naming and Directory Interface (JNDI) from other Java EE components but also from non-Java EE components. In this article we distinguish these two use-cases and provide a step-by-step guide for accessing EJBs deployed in the SAP NetWeaver Application Server, Java ™ EE 5 Edition, from non-Java EE clients.

Author: Vladimir Pavlov

Company: SAP Labs Bulgaria

Created on: 10 January 2007

Author Bio

Vladimir is a member of the SAP NetWeaver Application Server Java development team, focusing on the latest standards in the space of Java Platform, Enterprise Edition. He has been working for several years on application server implementation topics and application programming models.

Table of Contents

Applies to: 1
Summary 1
Author Bio 1
Table of Contents 2
Enterprise JavaBeans and EJB Clients 2
Creating and Implementing the EJB Application 3
Deploying the EJB Application 4
Creating and Implementing the EJB Client 5
  Analyzing the Client Code 7
Running the Client 7
Accessing EJB 2.x-style Home Interfaces 7
Conclusion 9
Related Content 9
Copyright 10

Enterprise JavaBeans and EJB Clients

It is well known and described in so many places (specifications, books, tutorials, and so on) how Enterprise JavaBeans (EJB) are accessed from other Java EE components. That is, the client component needs to declare a reference in its deployment descriptor (ejb-jar.xml, web.xml, application-client.xml) to the desired EJB and use its JNDI environment context (prefix *"java:comp/env/"*) to lookup the bean at runtime. EJB 3 and Java EE 5 make it even simpler for the application developer by introducing *dependency injection*. This allows you to annotate the corresponding field or property in your client code (which can be almost any Java EE component – EJB, servlet, JSF managed bean, application client, etc.) with the @EJB annotation and that's all - the container automatically "injects" the desired EJB reference so that you can use it in your business logic.

Unfortunately, the same is not true for non-Java EE clients. In this case there is no container to handle dependency injections and therefore the only means for gaining access to EJBs deployed in an application server remains the JNDI API. But what is even more frustrating for the EJB client developer is that there is no standard way for doing this across the different application server platforms, nor is there any hint about that in the specifications. That's why in this paper we'll provide a short guidance on how to access and call EJBs deployed in the SAP NetWeaver Application Server, Java EE 5 Edition, from non-Java EE clients. A similar article already exists for the NetWeaver 2004 and 2004s releases. It describes in details how to access the home interfaces of EJB 2.x beans and can be applied with almost no changes to the Java EE 5 Edition. Therefore we'll focus here on EJB 3 business interfaces and only mention the differences regarding EJB 2.x-style home interfaces.

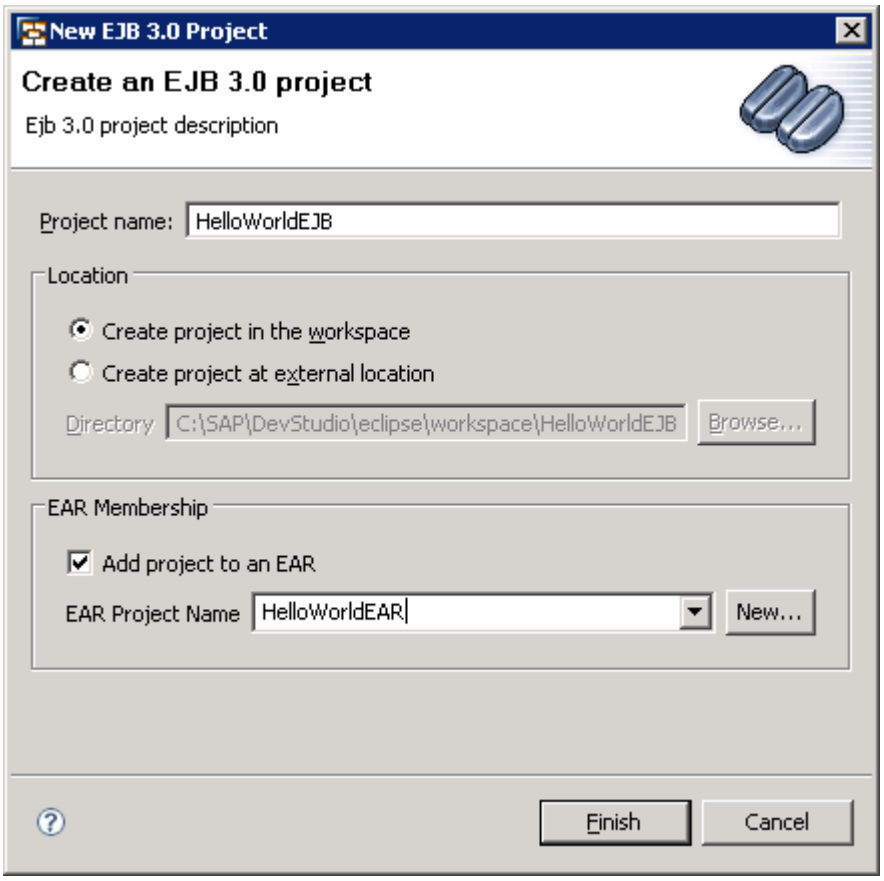
Please note that this article applies only to the SAP NetWeaver Application Server, Java ™ EE 5 Edition, and should not be thought of as a general guide for accessing EJBs using JNDI. The next major release of SAP NetWeaver Application Server will come with a detailed documentation and providing more powerful ways for executing EJB JNDI lookups.

To illustrate how EJBs can be accessed from non-Java EE clients, we'll first create a simple EJB application and deploy it on the Java EE 5 server. Then we'll write, setup, and run the client – a standalone Java program that connects remotely to the server and calls our session bean.

Creating and Implementing the EJB Application

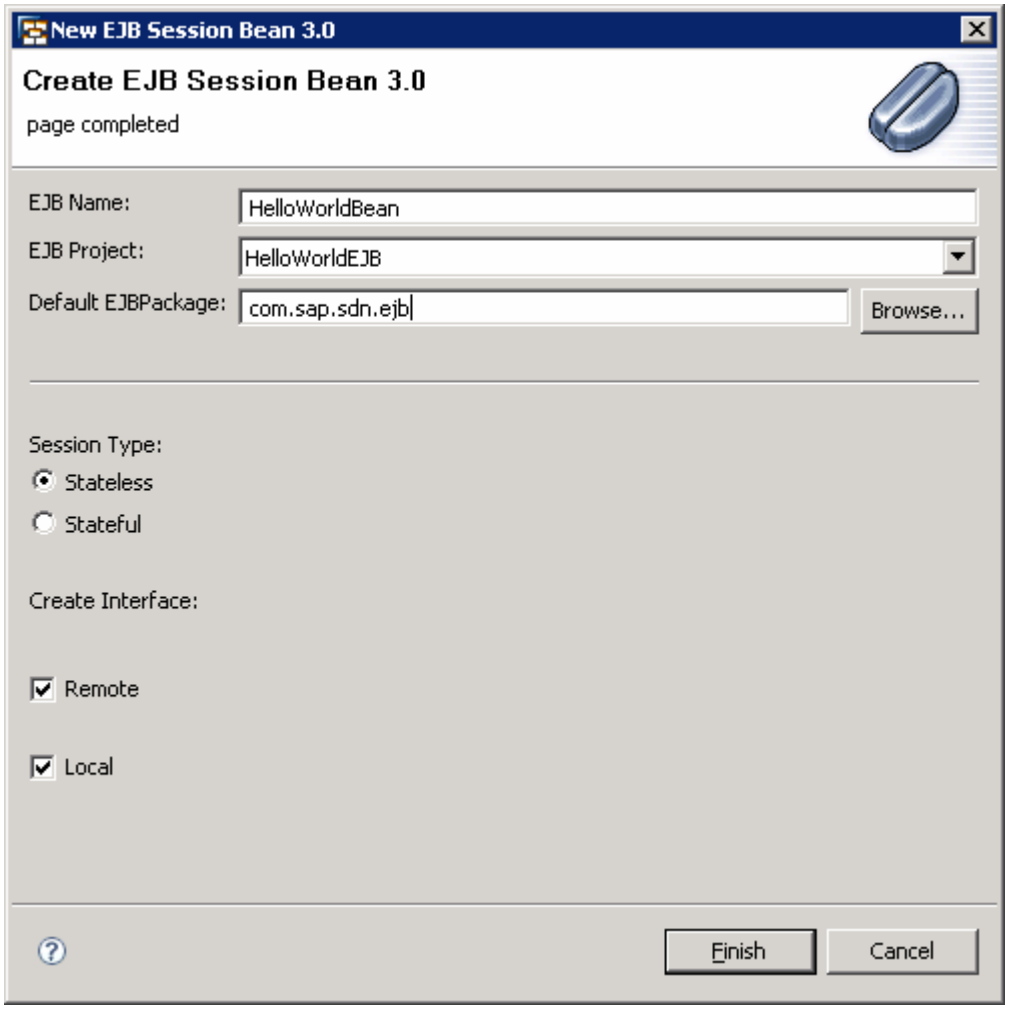
We'll use the *J2EE* perspective of the SAP NetWeaver Developer Studio for this step.

First, let's create a new EJB 3.0 project. Right-click in the *Project Explorer* and choose *New -> EJB Project 3.0*. Fill in the fields as shown in the figure below and choose *Finish*.



This also creates the *HelloWorldEAR* application project.

Now, we shall create and implement our session bean. From the context menu of the *HelloWorldEJB* project choose *New -> EJB Session Bean 3.0*. Enter the EJB name and package as *HelloWorldBean* and *com.sap.sdn.ejb*, respectively, and choose *Finish*.



Finally, let's implement some business method. It's quite simple and looks like this:

```
public String sayHello(String name) {
      return "Hello " + name;
}
```

Copy and paste the code above into the `HelloWorldBean` class. In the *Outline* view, right-click on the *sayHello* method and choose *EJB methods -> Add to Local and Remote Interfaces*. This will propagate the method to the business interfaces. Save the changes.

Deploying the EJB Application

Our EJB application is ready to be deployed.

Open the *Servers* view. From the context menu of the *SAP Server* node choose *Add and Remove Projects…* Add the *HelloWorldEAR* project to the Configured projects and choose *Finish*. If prompted, enter the user *Administrator* and the master password that you have supplied during the installation of SAP NetWeaver Application Server, Java ™ EE 5 Edition.

The application is deployed.

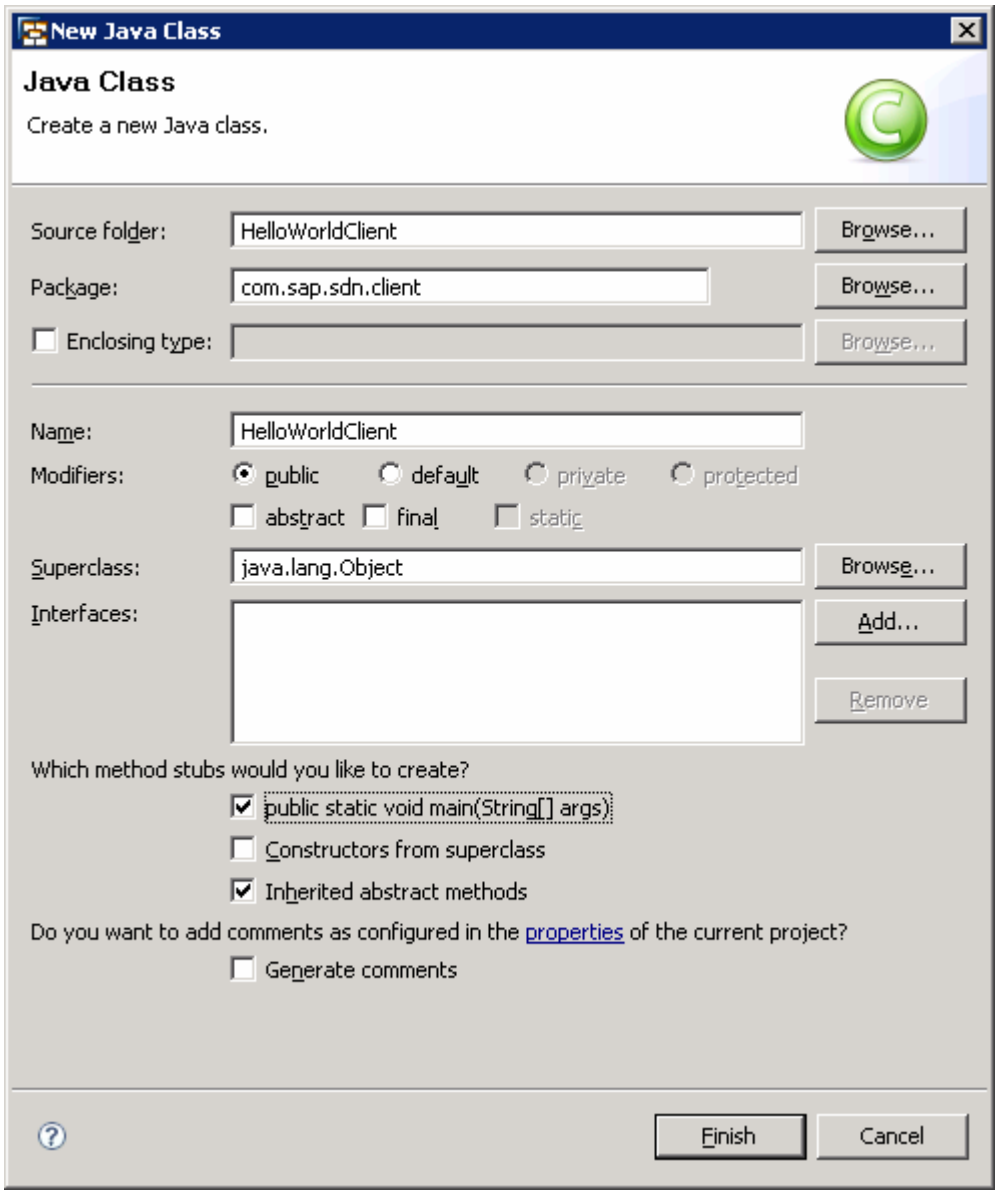
Creating and Implementing the EJB Client

For the implementation of the client – which as already outlined is a standalone Java program – we have to switch to the *Java* perspective of the SAP NetWeaver Developer Studio.

Right-click in the *Package Explorer* and choose *New -> Project… -> Java Project*. Enter the project name as *HelloWorldClient* and choose *Finish*.

In order to be able to use our EJB, we need a reference to the EJB project developed in the previous step. From the context menu of the *HelloWorldClient* project choose *Build Path -> Configure Build Path…* and go to the *Projects* tab. Add the *HelloWorldEJB* project and choose *OK*.

Now, let's implement the client. Create a new Java class in our client project and fill in the fields in the wizard as shown below.



Choose *Finish*.

The `HelloWorldClient` code is shown below:

```
package com.sap.sdn.client;

import java.util.Properties;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.rmi.PortableRemoteObject;

import com.sap.sdn.ejb.HelloWorldRemote;

public class HelloWorldClient {

      public static void main(String[] args) {
            Properties props = new Properties();
            props.put(Context.INITIAL_CONTEXT_FACTORY,
                  "com.sap.engine.services.jndi.InitialContextFactoryImpl");
            props.put(Context.PROVIDER_URL, "localhost:50004");

            try {
                  Context ctx = new InitialContext(props);
                  Object o = ctx.lookup(
"sap.com/HelloWorldEAR/REMOTE/HelloWorldBean/com.sap.sdn.ejb.HelloWorldRemote");
                  HelloWorldRemote helloRef = (HelloWorldRemote)
                        PortableRemoteObject.narrow(o, HelloWorldRemote.class);
                  String msg = helloRef.sayHello("Friend");
                  System.out.println(msg);
            } catch (Exception e) {
                  e.printStackTrace();
            }
      }

}
```

Analyzing the Client Code

First we initialize the properties used for obtaining the JNDI `InitialContext`. The first one is the initial context factory implementation class and the second one is the URL for connecting to the server. The latter is in the form <server_host>:<port>, where the port is constructed according to the pattern 5<JC_number>04. (On a default installation it's JC00, hence 50004.)

After that we connect to the server by creating a new `InitialContext` instance, and do the actual lookup of the EJB remote business interface. The most important thing to note here is the lookup string. Here is the scheme for its construction:

```
"<provider-name>/<app-name>/<access-type>/<ejb-name>/<interface-name>"

where:

<provider-name>   is the provider name as specified in the application-j2ee-
                  engine.xml, if any; defaults to sap.com;
<app-name>        is the name of the EAR application;
<access-type>     is either LOCAL or REMOTE depending on whether you are
                  accessing a local or remote business interface;
<ejb-name>        is the name of the bean;
<interface-name>  is the fully-qualified business interface name.
```

Of course, from a remote client we cannot access a local business interface. But if our client were running on the same server as the EJB and if it were not a standard Java EE client, we could use the similar string `"sap.com/HelloWorldEAR/LOCAL/HelloWorldBean/com.sap.sdn.ejb.HelloWorldLocal"` – just replacing the access type and interface name.

After the lookup we have to narrow the object to the actual interface (only necessary for remote interfaces) and then we are ready to call its business method. The result is printed on the standard output.

Running the Client

In order to be able to run the client, in addition to the EJB application classes we also need on the classpath the client classes of the Java EE 5 server. Those can be found in the *sap.com~tc~je~clientlib~impl.jar*, *sap.com~tc~exception~impl.jar*, and *sap.com~tc~logging~java~impl.jar* in the folder C:\SAP\JP1\JC00\j2ee\j2eeclient (on a default installation). Add these three jars to the build path of the *HelloWorldClient* project (from the context menu choose *Build Path -> Configure Build Path… -> Libraries tab -> Add External JARs…*).

Our EJB client is ready to be run. Right-click on the *HelloWorldClient.java* and choose *Run As -> Java Application*. You should see "`Hello Friend`" in the *Console* view.

Accessing EJB 2.x-style Home Interfaces

As already mentioned, there are some minor changes you have to make to your client if you want to access an EJB 2.x-style home interface (either of EJB 2.x or 3.0 beans).

First of all, let's add a remote home and component interfaces to our sample *HelloWorldBean*. Expand the *ejbModule* folder of the *HelloWorldEJB* project and create two new interfaces in the *com.sap.sdn.ejb* package: *HelloWorldHome* and *HelloWorld*. Below is the code for them:


```
package com.sap.sdn.ejb;

import java.rmi.RemoteException;

import javax.ejb.CreateException;
import javax.ejb.EJBHome;

public interface HelloWorldHome extends EJBHome {

      public HelloWorld create() throws CreateException, RemoteException;

}
```

```
package com.sap.sdn.ejb;

import java.rmi.RemoteException;

import javax.ejb.EJBObject;

public interface HelloWorld extends EJBObject {

      public String sayHello(String name) throws RemoteException;

}
```

Add the `@RemoteHome` annotation to the bean class to designate the home interface. It should look like this:

```
@Stateless
@RemoteHome(HelloWorldHome.class)
public class HelloWorldBean implements HelloWorldRemote, HelloWorldLocal {
```

Save the changes and open the *Servers* view. Redeploy the EJB application by selecting *Publish* from the context menu of the *SAP Server* node.

Now, let's modify the client to work with the newly added home and component interfaces. All we need to change are the following two lines of code (and adjust the import statements, of course):

```
                  Object o = ctx.lookup(
"sap.com/HelloWorldEAR/REMOTE/HelloWorldBean/com.sap.sdn.ejb.HelloWorldRemote");
                  HelloWorldRemote helloRef = (HelloWorldRemote)
```



```
                    PortableRemoteObject.narrow(o, HelloWorldRemote.class);
```

Replace them with:

```
                Object o = ctx.lookup("sap.com/HelloWorldEAR/HelloWorldBean");
                HelloWorldHome helloHome = (HelloWorldHome)
                      PortableRemoteObject.narrow(o, HelloWorldHome.class);
                HelloWorld helloRef = helloHome.create();
```

As you can see, the main difference here is in the string used to lookup the home interface. It is constructed according to the scheme:

“<provider-name>/<app-name>/<ejb-name>”

If our non-Java EE client were running on the server and were going to lookup the local home interface, we would use exactly the same lookup string, only with the prefix “`localejbs/`”, i.e.:

“localejbs/<provider-name>/<app-name>/<ejb-name>”

Finally, since `HelloWorldHome` and `HelloWorld` are referencing some classes from the `javax.ejb` package, in order to build and run the client we need also the *ejb-3_0-api.jar* on its classpath. It can be found in the folder C:\SAP\JP1\JC00\j2ee\cluster\bin\ext\ejb_api. Add it to the build path of the *HelloWorldClient* project as described in the section *Running the Client*.

That is it. We can now run our EJB client and again see the output in the *Console* view.

Conclusion

This article has provided you with the steps to access and call EJBs deployed in the SAP NetWeaver Application Server, Java EE 5 Edition, from non-Java EE clients. It hopefully demonstrated that creating and consuming especially EJB 3.0 business interfaces is really easy. You can reuse the sample client code as a template for your own projects and testing purposes.

Related Content

- SAP NetWeaver Application Server, Java EE 5 Edition
- Java Platform, Enterprise Edition 5 at SAP
- SDN Wiki Space Dedicated to Java EE 5 Development
- How To… EJB: Accessing EJB Applications Using JNDI
- Using Java EE Application Clients

Copyright